Bidirectionally Tolerating Inconsistency: Partial Transformations

Bidirectionally tolerating inconsistency: partial transformations

Perdita Stevens

School of Informatics
University of Edinburgh

Abstract. A foundational property of bidirectional transformations is that they should be correct: that is, the transformation should succeed in restoring consistency between any models it is given. In practice, however, transformation engines sometimes fail to restore consistency, e.g. because there is no consistent model to return, or because the tool is unable to select a best model to return from among equally good candidates. In this paper, we formalise properties that may nevertheless hold in such circumstances and discuss relationships and implications.

1 Introduction

In software engineering, it has long been understood that data involved in the development of a software system will, at least at certain points, be inconsistent, and that it can be unproductive – or even counterproductive – to try to maintain consistency at all times. Rather, mechanisms are needed to tolerate, manage and understand inconsistency so that it does not threaten the overall aim of the project [3, 13, 12]. Although consistency is a central notion for bidirectional transformations, tolerating inconsistency has so far not been studied in this context. This paper aims to rectify the situation.

In model-driven development, a bidirectional transformation (abbreviated bx) has two jobs. First, it has to be able to say when two models (or more: but in this paper, for clarity of exposition, we will treat only the case of two) are consistent according to the definition embodied in the transformation. Second, it has to be able to take a pair of models and modify a specified one of them so as to restore consistency. The question of what properties are desirable in such a transformation is an interesting one on which there is a growing literature. Most of it, however, does not allow the bx ever to fail, or to put it more positively, to succeed only partially. Consistency is all-or-nothing, and consistency restoration must always succeed for the bx to be considered *correct*.

Practical model transformation engines, by contrast, frequently fail to restore consistency between models when asked to do so. Although it is useful to try to develop languages and tools that will allow this to happen seldom, we suggest that there is also a need for a formal framework which recognises the inevitability of such failure. Without one, we cannot discuss and reason about the good properties that such "imperfect" bx may still have. Since a bx is usually assumed/required to restore perfect consistency, we could think of a bx that

does not as *partial*, its domain being the pairs of models on which it succeeds in restoring consistency. We shall, in fact, set up a more informative framework than this idea suggests, but, to motivate it, let us first consider three reasons why a bx engine may fail to restore consistency.

Let us assume that we are trying to check and, if necessary, restore consistency between a model m in a model space M and a model n in a model space N (for present purposes a model space may be thought of as just a set of models). In this paper we will consider *state-based* bx that modify just one of the two models, regarding the other as authoritative, so the bx has access to the two models, but not to information about what edits have been performed, traceability links, etc. (But see Section 6 for comment about ongoing work.)

First, it may be that the consistency relation itself is partial. For example, perhaps m and n are not fully consistent, we are asked to modify n to restore consistency, but there is no model $n' \in N$ that is fully consistent with m. This may happen for a number of reasons. For example, it may be that the notion of consistency being used imposes a condition on m which is currently not satisfied (say, "m and n are consistent if they both satisfy all their metamodel constraints, and ..." where m fails a metamodel constraint). In this case, since the bx has been asked to modify only n, regarding m as authoritative, it will not be able to succeed. What, then, should the tool do? Should it simply make no changes and report failure? This may be the only reasonable course (corresponding to a simple admission that (m, n) is not in the domain of this partial bx). On the other hand, perhaps there are certain changes that, even though they cannot completely restore consistency, bring the models closer to being consistent. We might prefer that the tool make these changes, leaving a "smaller" (in an appropriate sense) inconsistency that must be tolerated.

Second, it may be that there is a fully consistent model n', but the transformation engine, for some reason, may fail to return it. Perhaps we know as a matter of theory that a fully consistent model always exists, but finding it may require an infeasible amount of exploration and backtracking, so that a "quick and dirty" tool that achieves some improvement, leaving human intelligence to do the rest, is preferable. Another reason might be a desire to limit the kind of changes that the bx engine is allowed to make to a model. We see such a case in practice where the "models" are file systems and the "bx engine" is a file synchroniser. Typically, users are quite happy for the file synchroniser to propagate "non-conflicting" changes. However, where "conflicting" changes have been made to files on each side, users typically prefer to resolve conflicts manually. In this case, the consistency restoration that does not restore perfect consistency is actually preferred to one that did, because it is considered more trustworthy. A bx that does not have to resolve every problem and restore consistency perfectly may be faster, more understandable, easier to verify and generally safer than one that restores perfect consistency at all costs.

Third, it may be that there is more than one consistent model that the bx tool could return, but the transformation does not provide information that would allow the tool to choose between them, either through oversight or because the

best choice depends on factors that are not formalisable. There are then three options, any of which may be preferred depending on the circumstances. The first two are variants of non-deterministic (but total) transformations. The tool might make an arbitrary (or heuristically guided) choice of consistent model, e.g., by returning the first one that is found by some search algorithm. This may be fine if the differences between the different consistent models are really arbitrary, i.e. of no interest to users of either model. The tool may find all consistent models, and return them, or a concise description of them, to the user for the user to pick one, with the transformation not being considered complete until the choice has been made. This is difficult to make practical, tending to be expensive both in computational and human effort terms. In this paper we wish to consider a third option: that the tool returns a model n' which is not fully consistent with m, but which is, in a sense to be formalised, more consistent with m than n was, incorporating only "uncontroversial" modifications. For example, n' may add model elements that are common to all fully consistent models, but not those that are present in only some fully consistent models. The user may modify n' further to make it fully consistent at a later date, but need not do so immediately. (Perhaps, even, a later change to m and a later application of a bx may obviate the need to edit the model on the N side manually at all.)

In each case, it is desirable to have a framework in which we can unambiguously state the properties that a bx has or should have, so that ultimately these properties can be verified and guaranteed, granting the developer greater confidence about what applying the bx will do.

Such a framework also facilitates modular development of, and reasoning about, total bx, which can now be built by chaining partial bx. This lets us separate concerns, e.g., let different partial bxes fix different parts of a model, if these are sufficiently independent. Or we may let one partial bx delete elements, and another add. We say more about this in Section 4. To talk about the properties of the parts and about their composition, we need to make explicit the existence of situations that are in-between perfect consistency and "all bets are off" inconsistency.

The remainder of this paper is structured as follows. In Section 2 we discuss related work. In Section 3 we introduce some fundamental definitions. In Section 4 we illustrate these with examples. Section 5 discusses how different partial bidirectional transformations, involving the same or different model spaces, may be related. In Section 6 we conclude and briefly discuss further work to be done.

2 Related work

Much of the work on bx is done with the special kind of bidirectional transformations that are lenses[4], in which one model space (the *view*) is a strict abstraction of the other (the *source*). In this case[1] $\overrightarrow{R}$ is replaced by `get`, a one-argument function from concrete source to abstract view, and $\overleftarrow{R}$ by `put`, a

[1] forward reference to notation defined in Sect 3

two-argument function which, given an updated view and a source, produces an updated source.

Diskin's seminal paper [7], which is not limited to the lens case, already briefly considered partial bidirectional transformations. He points out that in the lens setting, partiality of `put` follows from the lens laws if `get` is allowed to be non-surjective. This paper did not, however, model different levels of consistency; as far as the author is aware, that is being done for the first time here.

Indeed, the lens approach to bidirectionality described in e.g. [9, 4] arose, historically, from earlier work relating to file synchronisation [2]. This gave a precise semantics to a file synchroniser, in the process discussing properties that a synchroniser should have. It made a clean separation between update detection and conflict resolution, formalising the slogan "non-conflicting updates are propagated". Propagating non-conflicting updates, while leaving conflicts for a user to resolve manually, is an example of the kind of scenario we deal with here.

Another, more recently active, strand of related work is that in bidirectionalization [17]. This, too, applies in the lens special case of bidirectional transformations. Its premise is that a `get` function is given, and a `put` function must be inferred from it. Of course, sometimes it is not possible to determine a reasonable `put` function, while sometimes many different `put` functions are possible; part of the interest of this problem area is to investigate the situation. The work aims to determine under what circumstances it is possible – and then how – either to determine the "best" `put` function in an appropriate sense, or to give a choice of good `put` functions to the transformation developer in a reasonable way.

The connection between bidirectionalization and the present work is as follows. In the lens scenario, giving a `get` function is equivalent to giving a consistency relation; m is consistent with n iff $n = get(m)$. Key to [17] is a distinction between shape-preserving and shape-changing updates to a model. In the list case considered, the shape of a model is its length, while its content is the actual list elements. The bx is assumed to respect the division into shape and content in that, for example, the results of applying `get` to two lists having the same shape will be two lists that also have the same shape (as one another, not necessarily as the inputs). This enables syntactic bidirectionalization to be used as a plug-in for semantic bidirectionalization: the bidirectionalization problem is factored into how to deal with shape and how to deal with content. One can, for example, produce a `put` function that works only in the case that the shapes of its arguments are appropriately related. This has the same flavour as our concerns here: we wish to support bx tools that can do the right thing in straightforward cases, with a clear specification of what that means, even if they cannot succeed in all cases. More formally, we may model the bidirectionalization case using a consistency structure with three possible values: inconsistent $<$ shape-consistent $<$ perfectly consistent. Then we may separate two actions: first, the process of making an update that increases consistency to shape-consistent; second, the process of making an update that achieves perfect consistency, provided that shape-consistency is given to begin with. This separation of concerns may make it easier to identify the design choices. It may also allow us to build a bx out of

components, each able to do one of these two actions. Even if this is not possible, there may be value in automating one of the actions.

In the database literature, consideration of one aspect of partial bidirectional transformation goes back at least to [10, 11]. This is the issue of how the user of a view may limit updates on the view to the admissible ones, that is, may stay within the domain of an automatically definable `put` function. More recently, study of related problems has been driven by the need to query, update and repair databases that contain errors or omissions, leading them to violate integrity constraints. A seminal paper in this area was [1], which discusses how to produce, in response to a query, all the tuples that would have to be present in a response to its being made against any repair of the actual database to one that satisfied its integrity constraints. One may normally see a query answer as a model which is consistent with a database, with respect to a particular query, when it is the answer to that query on the database. In this case, however, the ideally desired query answer is actually what would be returned from the ideal repaired database (which does not, in fact, exist). Because it does not exist, the best that can be done is an approximation; the answer set here can be seen as one that is partially consistent, having made (to the empty model) only the uncontroversial changes (adding the tuples that will definitely be present).

We note in passing that the promisingly named [8] is not about partial (model transformations) but about (partial model) transformations and though interesting is not closely related.

3 Basic definitions

We begin by presenting a framework in which we generalise the possible results of a consistency check from true/false to a more nuanced judgement.

Let $(\Lambda, \leq_\Lambda)$ (which we will normally refer to just as Λ) be a partially ordered set, the *consistency structure*, having a top element $\top$ ("perfectly consistent"; all $\lambda \leq_\Lambda \top$). This will often be a lattice.

Let a partial bidirectional transformation $R : M \leftrightarrow N$ over Λ be defined by specifying

- a consistency indicator $R : M \times N \to \Lambda$ that says how consistent a given pair of models is
- consistency restoration functions $\overrightarrow{R} : M \times N \to N$ and $\overleftarrow{R} : M \times N \to M$.

From now on this is what we shall mean by a bidirectional transformation (bx) – when we want to use the usual definition in which the consistency indicator is a relation, we shall talk about *total bx*. A total bx may be represented as a special case of a partial bx, over the two element lattice with elements $\top$ (consistent) and $\bot < \top$ (inconsistent).

Note that the restoration functions of a partial bx are still total, but, as we shall see, they may not succeed in restoring consistency. In the worst case (corresponding intuitively to a partial bx being invoked outside its domain) it is always open to the function to return the argument with appropriate type,

i.e. to propose no change. Thus making these total functions is no restriction. Of course, those definitions of properties of bxs that refer only to the consistency restoration functions do not need to be altered in this setting, though they may need reinvestigation. An example (following the terminology of [7]) is

Definition 1. *A bx $R : M \leftrightarrow N$ is* history ignorant *if for all $m, m' \in M$ and $n \in N$, we have $\overrightarrow{R}(m, \overrightarrow{R}(m', n)) = \overrightarrow{R}(m, n)$ and dually.*

Other standard definitions need minor adjustments to their notation, to make them apply to partial bx:

Definition 2. *A bx R is* consistency-total *if for any m there is some n such that $R(m, n) = \top$, and dually.*

Definition 3. *A bx R is* correct *if for all $m \in M$ and $n \in N$ we have $R(m, \overrightarrow{R}(m, n)) = \top$, and dually.*

Definition 4. *A bx R is* hippocratic *if for all $m \in M$ and $n \in N$ we have $R(m, n) = \top \Rightarrow \overrightarrow{R}(m, n) = n$, and dually.*

We can now define

Definition 5. *A bx R is* improving *if it always returns a model that is at least as consistent as its argument was. That is, $R(m, \overrightarrow{R}(m, n)) \geq_\Lambda R(m, n)$, and dually.*

An improving bx is allowed to return something that is no more consistent than its argument, but yet is not identical to it. Whether it is desirable to permit this depends on circumstance. If a model is an XML file, not intended for human reading, then changing things that are not important to consistency may be perfectly acceptable. Modelling tools often silently change the identifiers of model elements, for example. On the other hand, at times it is essential that users can be confident their models are not changed unnecessarily. For example, the layout of a diagram is typically irrelevant to consistency and indeed semantics, yet users of diagrams intensely dislike their layouts being changed. So that we can talk about this, we define

Definition 6. *A bx R is* as hippocratic as possible *(AHAP) if its consistency restoration functions return exactly their argument of appropriate type, unless returning something strictly more consistent. That is,*

$$\overrightarrow{R}(m, n) = n \ \vee \ R(m, \overrightarrow{R}(m, n)) > R(m, n)$$

and dually.

Note the need for this definition to make use of *strict* increase in Λ. Intuitively, properties that do this tend to be harder to work with than those that can be expressed just with the partial order's $\leq$.

This begins to let us specify what damage a partial bx satisfying certain properties may *not* do; next we turn to how to ensure that it *does* achieve what it should.

Definition 7. *Given $m \in M$, the set of* $\overrightarrow{R}$ candidates with respect to m *is* $\{n' \in N : R(m,n')$ *is maximal*$\}$. *That is, n' is a candidate iff $R(m,n') = \lambda \in \Lambda$ such that there does not exist any $n'' \in N$ with $R(m,n'') >_\Lambda \lambda$. If a candidate n' further satisfies $\forall n'' \in N.R(m,n') \geq_\Lambda R(m,n'')$ then it is* dominant. *Dually for* $\overleftarrow{R}$.

We normally abbreviate and say n is a (dominant) candidate wrt m. Notice that n being a candidate wrt m does not in general imply that m is a candidate wrt n. It turns out to be theoretically convenient if it does, though:

Definition 8. *A bx is* balanced *if for all $m \in M$ and $n \in N$, m is a candidate wrt n iff n is a candidate wrt m.*

In particular this will be the case if the bx is consistency-total. Otherwise, we obviously cannot expect a bx to be correct, but the next best thing is:

Definition 9. *A bx R is* as correct as possible *(ACAP) if it always returns a candidate.*

An ACAP bx must not return something if there is something strictly better it could have returned instead. In particular if, for given $m \in M$, a dominant $\overrightarrow{R}$ candidate exists, an ACAP bx must return one. If the consistency structure is a total order, all candidates are dominant. Then the consistency *level* λ achieved when $\overrightarrow{R}$ or $\overleftarrow{R}$ is applied to (m,n) is determined by (m,n) and the consistency indicator R, even though R does not determine $\overrightarrow{R}$, $\overleftarrow{R}$.

Let us collect some immediate consequences of the definitions.

Proposition 1. *1. If R is correct, then it is consistency-total and ACAP.*
2. *If R is AHAP, then it is hippocratic.*
3. *If R is correct, then it is hippocratic if and only if it is AHAP.*
4. *If R is consistency-total, then it is correct if and only if it is ACAP.*
5. *If R is consistency-total then R is balanced; the candidates and the dominant candidates (wrt m) are both just the set of perfectly-consistent elements (wrt m).*
6. *If R is either AHAP or ACAP, then it is improving.*

3.1 Subspaces and subspace pairs

A natural partner of the idea that some model pairs are more consistent than others is the idea that some regions of a pair of model spaces are more compatible than others. It turns out that the connection is more than just intuitive.

Definition 10. *Let $M' \subseteq M$ and $N' \subseteq N$ and let $R : M \leftrightarrow N$ be a bx (partial or total). We say (M', N') is a* subspace pair *if $\forall m \in M'.\forall n \in N'.\overrightarrow{R}(m,n) \in N' \wedge \overleftarrow{R}(m,n) \in M'$. We say M' is a* subspace*, and write $M' \trianglelefteq M$, if (M', N) is a subspace pair. (We define $N' \trianglelefteq N$ dually.)*

A subspace pair is a place where two teams of developers, each modifying a model that may be synchronised by a bx, may agree to stay. If (M', N') is a subspace pair with respect to R, then provided neither team moves their model outside M', N' respectively, the transformation will never move them outside. A subspace is a place where one team may unilaterally decide to stay.

It often happens, in MDD, that there are properties of a model that are ultimately desirable, but not enforced by the tool in which the model is developed – maybe not even desired at some stages of development. Strict compliance with metamodel constraints is an example. During development we may need the bx to propagate changes even though the models are not metamodel-compliant, say; the most helpful bx may also guarantee that, provided the human developers do not break compliance, the transformation will not. The tool may do what it can on non-compliant models, but it is natural to think that models that are also compliant are more consistent than those that are not. That is, the metamodel-compliant model pairs form a subspace pair within which a higher consistency value is obtainable than outside. In Section 5 we shall see how to relate bx on a subspace pair to bx on the whole model spaces. For now:

Lemma 1. *Let $R : M \leftrightarrow N$ be a partial bx over Λ.*

1. *M, N are subspaces. Unions and intersections of subspaces are subspaces.*
2. *If M_1 and N_1 are subspaces, then (M_1, N_1) is a subspace pair. For example, (M, N) is a subspace pair.*
3. *If (M_i, N_i) is a subspace pair for each $i \in I$ an index set (of any cardinality), then $(\bigcap_{i\in I} M_i, \bigcap_{i\in I} N_i)$ is a subspace pair.*
4. *If R is AHAP, then for any $m \in M$ and $n \in N$ such that each is a candidate with respect to the other (e.g., $R(m, n) = \top$), $(\{m\}, \{n\})$ is a subspace pair.*
5. *Let $M_\top = \{m \in M : \exists n \in N s.t. R(m, n) = \top\}$ and define $N_\top$ similarly. If R is ACAP, then $(M_\top, N_\top)$ is a subspace pair.*
6. *If R is ACAP and the consistency structure Λ is a total order, then for each $\lambda \in \Lambda$ we define $M_\lambda = \{m \in M : \exists n \in N s.t. R(m, n) \geq_\Lambda \lambda\}$ and N_λ similarly. Then (M_λ, N_λ) is a subspace pair.*

Of course, in general the component-wise union of two subspace pairs is not one.

4 Examples

In this section we set up a collection of examples, briefly described, to illustrate the definitions already given and what follows.

4.1 Families of trivial examples to illustrate definitions

1. Let Λ be the one-point lattice whose only point is $\top$. Then any bx over Λ is consistency-total, balanced, correct, ACAP and improving, but generally not AHAP or hippocratic.

2. Let M and N be any sets, let Λ be any partial order, and let $R : M \times N \to \Lambda$ be any function, serving as consistency indicator.
   (a) Let $\overrightarrow{R}$ be the second projection, that is, $\overrightarrow{R}(m, n) = n$ for any m and n, and let $\overleftarrow{R}$ be first projection. Then this bx – which makes no attempt ever to restore consistency – is hippocratic, AHAP and improving. In general it will not, of course, be correct or ACAP.
   (b) Alternatively, let $\overrightarrow{R}$ be defined by $\overrightarrow{R}(m, n) = n$ if $R(m, n) = \top$, otherwise $\overrightarrow{R}(m, n) = \Omega_N$, the special "no information" or empty model, which is perfectly consistent with the corresponding empty model in M, Ω_M, only. That is, suppose that $\overrightarrow{R}$ deletes everything if it finds anything other than perfect consistency. Then R is hippocratic, but does not in general have any of our other properties.

4.2 Composer examples

For ease of presentation, our remaining examples of partial bx will be variants on the (total) COMPOSERS example [16, 6], illustrated in Fig. 1. Until we introduce a slight variation later, our bxes will use the same pair of model spaces as COMPOSER. A model $m \in M$ comprises a set of (unrelated) Composer objects, each with a name, dates and nationality. A model $n \in N$ is an ordered list of pairs each giving a name and a nationality.

Now we consider different ways to define consistency and consistency restoration. We will refer to the consistency partial orders shown in Fig. 2.

Consistency as a conjunction To be perfectly consistent, models may have to satisfy several conditions, which need not be ranked. In QVT-R [14], for example, consistency is specified as a conjunction of two directional checks. That is, to check whether m is consistent with n, we must check whether m is acceptable from the point of view of n, and vice versa; m and n are considered consistent iff both of these checks succeed. The most faithful QVT-R tool, TATA's ModelMorf, actually requires the two directions to be checked separately by the user, even though the QVT-R standard's notion of consistency is the conjunction of the two results.

We may model such a situation using the consistency structure Λ_{LR} from Fig. 2. Following COMPOSERS [16], our composer models (say (m, n)) are deemed to be perfectly consistent ($R(m, n) = \top$) iff both: for every composer in m, there is at least one entry in the table comprising n with the same name and nationality; and, for every composer in n, there is at least one entry in m with the same name and nationality. If only the first condition holds, let $R(m, n) = \lambda_L$, if only the second, let $R(m, n) = \lambda_R$, if neither holds, let $R(m, n) = \bot$.

Given this notion of consistency as part of a partial bx over Λ_{LR}, we still have a choice about the consistency restoration functions. One user might be happy with automatically deleting composers, but dislike the "kludge" of making dates "????-????" and prefer manually inserting new composers to making the

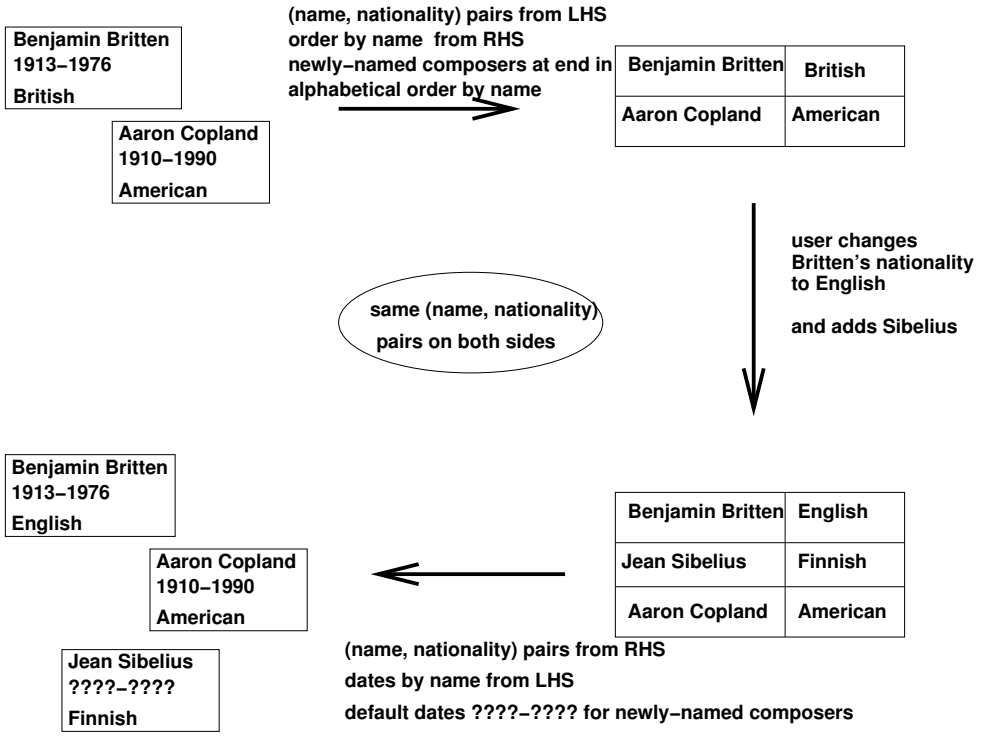

Fig. 1. Two kinds of composer model, illustrating a total bx

bx, in effect, choose among consistent models (the dates being irrelevant to consistency). Another user might be happier about elements being automatically added than about them being deleted. Then a suitable option is to make $\overrightarrow{R}$ add to n any missing composers – thus ensuring that consistency reaches at least λ_L – but not to make it delete any composers who do not occur in m. Together with the dual behaviour for $\overleftarrow{R}$, this gives a partial bx that improves consistency, and will not make any change except to do so, but does not guarantee to make the models completely consistent. That is, it is improving, indeed AHAP, but not correct or ACAP. (The same can be made true of the first user's preferred partial bx.)

The discipline for using such a bx manually must be considered carefully. Suppose the $\overrightarrow{R}$ just discussed is applied, and does not restore perfect consistency because there is a composer in n but not m. Now if $\overleftarrow{R}$ is immediately applied – without the extra composer being manually deleted – this composer will be re-added to m.

A different scenario is where such a bx is used as a phase within a total bx; then this discipline is in effect provided automatically. In QVT-R, which does not discuss partial consistency, ensuring that both checks succeed while modifying

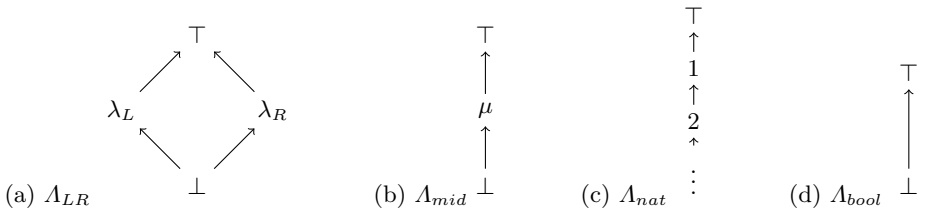

Fig. 2. A few consistency partial orders

only one of the models motivates the two-phase enforce process. When n must be modified so that it is consistent with fixed m, the first phase adds model elements that are required to exist in order for n to become acceptable from the point of view of m; the second, delete, phase removes model elements from n that cannot be present if m is to be acceptable from the point of view of n. In fact, making this work is far from trivial and is the source of some confusion concerning the semantics of QVT-R [5]. Separating the semantics of such bx explicitly into phases each described by a partial bx may be useful.

Consistency levels as diagnostics Even if we have an ordinary total bx that is correct and hippocratic, we may prefer something that gives more information, for engineering reasons. That is, we might like to be told not only that a given pair of models is inconsistent, but something about why they are inconsistent. If we want our framework to model this, it can do so by means of a partial bx that corresponds to the original in the sense that the perfectly-consistent pairs are the same in both cases, but the partial bx has, instead of a single judgement $\bot$, "inconsistent", a collection of possible consistency levels. Both bx may have the same consistency restoration functions, so the partial bx will, like the original it is based on, be correct and hippocratic (hence ACAP, AHAP and improving).

To give a simple example, we might use the natural numbers as consistency levels to indicate the number of errors counted in some way. In our example, this could be the number of composers who occur on one side but not the other. We could define a bx whose consistency restoration was just like the total bx illustrated in Fig. 1, but over the consistency structure Λ_{nat} from Fig. 2.

Separation of consistency concerns An example somewhat reminiscent of those used in the bidirectionalization work discussed in Section 2 is the following.

Consider bx between M and N over consistency structure Λ_{mid} from Fig. 2. Let the middle consistency element, μ, indicate that the same names occur in each model. Let perfect consistency, $\top$, as before indicate that the same (name,nationality) pairs occur on both sides. This allows us to separate the task of designing consistency restoration functions, if we like, into the two parts of making sure the right names are present, and making sure the right nationality/ies also occur with each name.

A non-consistency-total example So far all our examples have been consistency-total. Now suppose we have the usual "same (name, nationality) pairs" requirement for consistency, but we also have a constraint (not enforced, but desirable) that composer names in a table in $n \in N$ must be no more than 16 characters long (while those in m are still unrestricted). We may model this using a bx over Λ_{LR} from Fig. 2. Suppose λ_L means both models comply with their constraints, λ_R means the same (name, nationality) pairs occur in both models, $\bot$ means neither holds and $\top$ means both hold.

The models illustrated in Fig. 1 are unaffected, but now consider $m \in M$ that includes a composer with name Pyotr Ilyich Tchaikovsky (and no other composers, for simplicity). Now there is no model $n \in N$ which is perfectly consistent with m. A bx that is asked to modify n must choose: either it can include this name, which will allow it to achieve consistency level λ_R, or it can comply with its constraints, achieving λ_L, but it cannot do both.

Suppose the bx writer settles on preferring metamodel compliance, and writes a bx that we shall call T as we shall refer to it again in Section 5.

Note that T is not balanced. Say we have m as above and a model n whose single row reads PI Tchaikovsky. Because n cannot be modified so that its composer name matches m's without violating the constraint, n is a candidate with respect to m. (It is not a dominant candidate, because the alternative choice of making the names the same but violating the constraint would have given an incomparable consistency value.) On the other hand, m could if we wish be modified so that its composer name matched n's, which would restore perfect consistency, so m is not a candidate with respect to n.

5 Relating partial bx

A major motivation for studying bx that tolerate inconsistency is to support the use of bx during development. At some stages we may have a well-developed notion of perfect consistency, yet not be in a position to enforce it immediately (e.g. because a model is currently incomplete, awaiting more information; it might even be syntactically incorrect). We do not want the inability to enforce perfect consistency to stop us synchronising models at all; we would like to be able to run bx and get some guarantees about what a bx will do. We also want to be able to give more information about the nature of the inconsistency discovered by a check. Later in development, a total bx may be usable, and we would like to know it is compatible in an appropriate sense with the partial bx used earlier.

This motivates the study of relationships between different partial bx. Many interesting examples relate bx that relate the same sets of models (perhaps restricting to a subset) but we start with a more general notion:

Proposition 2. *Let $R_1 : M_1 \leftrightarrow N_1$ be a partial bx on Λ_1. Let $f_\Lambda : \Lambda_1 \to \Lambda_2$ be a total function preserving the partial order, that is, satisfying $x \leq y \Rightarrow f_\Lambda x \leq f_\Lambda y$. Let $f_M : M_1 \to M_2$ and $f_N : N_1 \to N_2$ be surjective partial functions satisfying the following coherence condition: if $f_M m = f_M m'$ and $f_N n = f_N n'$ (in particular, m, m', n, n' are in the appropriate domains) then*

- $f_\Lambda R_1(m,n) = f_\Lambda R_1(m',n')$;
- f_N *is defined on* $\overrightarrow{R_1}(m,n)$, *and dually;*
- $f_N\overrightarrow{R_1}(m,n) = f_N\overrightarrow{R_1}(m',n')$, *and dually.*

Then the following gives a well-defined partial bx $R_2 : M_2 \leftrightarrow N_2$ *on* Λ_2, *which by slight abuse of notation we will denote* $f(R_1)$*:*

- $R_2(f_M m, f_N n) = f_\Lambda R_1(m,n)$
- $\overrightarrow{R_2}(f_M m, f_N n) = f_N\overrightarrow{R_1}(m,n)$ *and dually.*

In particular, notice that if $M_1 = M_2$ and $N_1 = N_2$, with f_M and f_N being total identity functions, the coherence conditions become trivial. That is, given a bx $R : M \leftrightarrow N$ over Λ_1, and a partial order preserving function $f_\Lambda : \Lambda_1 \to \Lambda_2$, we can always build a bx $f(R) : M \leftrightarrow N$ over Λ_2.

Of course the proof of Prop. 2 is easy: the coherence conditions are exactly what is needed. More interesting is to see what happens to properties of R_1.

Proposition 3. *If* R_1 *is improving then so is* $f(R_1)$.

To go further we need to impose further conditions on f_Λ, involving when one consistency value is *strictly* greater than another:

Proposition 4. *If* R_1 *is ACAP and, in addition to the conditions of Prop. 2,* f_Λ *satisfies* $f_\Lambda x > f_\Lambda y \Rightarrow x > y$, *then* $f(R_1)$ *is ACAP.*

The condition is necessary in order to handle situations like the composer example T on Λ_{LR}, in which consistency means the conjunction of two properties that are considered incomparable and cannot always both be achieved. Now suppose that f_Λ privileges one property over the other, so that ensuring that one is "better". In our example, suppose we consider $f_\Lambda : \Lambda_{LR} \to \Lambda_{mid}$ sending λ_L to $\perp$ and λ_R to μ; that is, f_Λ models that we now consider having the same (name, nationality) pairs to be better than obeying the constraints, in cases where it's not possible to do both. Our bx T over Λ_{LR} was ACAP, but $f(T)$ over Λ_{mid} is not. The condition captures the idea that turning incomparability into dominance in the consistency structure may break ACAP.

The next result might be considered dual:

Proposition 5. *If* R_1 *is AHAP and, in addition to the conditions of Prop. 2,* f_Λ *satisfies* $x > y \Rightarrow f_\Lambda x > f_\Lambda y$, *then* $f(R_1)$ *is AHAP.*

The condition is necessary because without it, R_1 may make a consistency improvement that is "erased" by f_Λ. That is, we may have $R_1(m, \overrightarrow{R_1}(m,n)) > R_1(m,n)$, but $f_\Lambda(R_1(m, \overrightarrow{R_1}(m,n))) = f_\Lambda(R_1(m,n))$. Unless the difference between n and $\overrightarrow{R_1}(m,n)$ is likewise erased by f_N, $f(R_1)$ will fail AHAP.

5.1 Example: restricting to a subspace pair

Let (M_2, N_2) be a subspace pair in (M_1, N_1) and let f_M, f_N be the identity on M_2, N_2, undefined elsewhere. Then as these are also injective where defined, the coherence conditions in Prop. 2 hold for any f_Λ preserving the partial order. If we take f_Λ to be the identity, what we get is just the restriction of a bx to a subspace pair. The conditions of Prop. 4 and Prop. 5 both hold, so we see that the restriction will be ACAP and AHAP if the original was.

Next, let us use Prop. 1 to consider the subspace pair $(M_\top, N_\top)$ in (M_1, N_1), and consider $f_\Lambda : \Lambda_1 \to \{\top, \bot\}$ given by $f_\Lambda(\lambda) = \top$ if $\lambda = \top$, otherwise $\bot$. We see that we can construct a total bx from any ACAP partial bx by, intuitively, throwing out all the elements on which consistency cannot be restored. The Prop. 4 condition holds, and since the consistency relation is, by construction, total on $(M_\top, N_\top)$ our restricted bx is correct. The Prop. 5 condition fails, but in this particular case we have thrown out all potential counterexamples, so:

Proposition 6. *Let $R : M \leftrightarrow N$ be a partial bx which is ACAP and AHAP. Define $R_\top : M_\top \leftrightarrow N_\top$ by:*

- *$M_\top = \{m \in M : \exists n \in N. R(m, n) = \top\}$, and $N_\top$ dually;*
- *$R_\top(m, n)$ holds iff $R(m, n) = \top$;*
- *$\overrightarrow{R_\top}(m, n) = \overrightarrow{R}(m, n)$ and dually.*

Then $R_\top$ is a correct and hippocratic total bx.

6 Conclusions and future work

In this paper we have begun a study of partial bidirectional transformations, being a generalisation of bidirectional transformations in which the consistency definition, rather than being a relation, is a function taking values in a partially ordered structure. We have given examples to show the practical potential of such bx, and have established a framework in which to discuss their properties and to understand the relationships between them and their properties.

Beyond this paper, we have begun a study of the implication, for this framework, of considering the edit monoid on each model space. This is promising from the point of view of least change: intuitively, difficulties arise when the edit path towards a candidate must involve going via a model that is less consistent than the original. It is good if edit paths in the model spaces project onto paths in the consistency structure.

More generally, we are studying the many ways in which bx formalisms go beyond the state-based approach we work with here, to include extra information e.g. edits, traces or an archive [9]. We are also studying the properties of bidirectional transformations that are related for total bidirectional transformations in [15], to see what happens when we transfer them to the partial setting. Do the same relationships hold between the notions? Are other special properties more important in this setting? Balanced bx are much more tractable than others, as they allow foundational results about the equivalences used in [15] to carry over.

Acknowledgements I thank James McKinna for very helpful conversations. I thank the referees for their constructive suggestions, including some that could not be implemented in this version for space reasons. The work is partly supported by EPSRC grant EP/K020218/1.

References

1. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. PODS*, pages 68–79. ACM Press, 1999.
2. S. Balasubramaniam and Benjamin C. Pierce. What is a file synchronizer? In *Proceedings of MobiCom'98)*, October 1998.
3. Robert Balzer. Tolerating inconsistency. In *Proceedings of ICSE'91*, pages 158–165. IEEE Computer Society / ACM Press, 1991.
4. Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: Resourceful lenses for string data. In *Proceedings of POPL'08*, January 2008.
5. Julian C. Bradfield and Perdita Stevens. Enforcing QVT-R with mu-calculus and games. In *Proceedings of FASE'13*, volume 7793 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2013.
6. James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. Towards a repository of bx examples. In *Proceedings of Bx'14*, 2014.
7. Zinovy Diskin. Algebraic models for bidirectional model synchronization. In *Proceedings of MODELS'08*, volume 5301 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2008.
8. M. Famelis, R. Salay, and M. Chechik. The semantics of partial model transformations. In *Proceedings of ICSE workshop on Modeling in Software Engineering*, pages 64 –69, June 2012.
9. J. Nathan Foster, Michael B. Greenwald, Christian Kirkegaard, Benjamin C. Pierce, and Alan Schmitt. Schema-directed data synchronization. Technical Report MS-CIS-05-02, University of Pennsylvania, March 2005.
10. Georg Gottlob, Paolo Paolini, and Roberto Zicari. Properties and update semantics of consistent views. *ACM Trans. Database Syst.*, 13(4):486–524, 1988.
11. Stephen J. Hegner. Foundations of canonical update support for closed database views. In *Proceedings of ICDT'90*, volume 470 of *Lecture Notes in Computer Science*, pages 422–436. Springer, 1990.
12. Alexander Nöhrer, Armin Biere, and Alexander Egyed. A comparison of strategies for tolerating inconsistencies during decision-making. In *16th International Software Product Line Conference, SPLC '12*, pages 11–20. ACM, 2012.
13. Bashar Nuseibeh, Steve M. Easterbrook, and Alessandra Russo. Leveraging inconsistency in software development. *IEEE Computer*, 33(4):24–29, 2000.
14. OMG. MOF2.0 query/view/transformation (QVT) version 1.1. OMG document formal/2009-12-05, 2009. available from www.omg.org.
15. Perdita Stevens. Observations relating to the equivalences induced on model sets by bidirectional transformations. *EC-EASST*, 049, 2012.
16. Perdita Stevens, James McKinna, and James Cheney. Composers v0.1 in Bx Examples Repository. http://bx-community.wikidot.com/examples:home. Date retrieved: 16 Jan 2014.
17. Janis Voigtländer, Zhenjiang Hu, Kazutaka Matsuda, and Meng Wang. Combining syntactic and semantic bidirectionalization. In *Proc. ICFP*, pages 181–192. ACM, 2010.